Package 'RDeco'

November 30, 2016

Type Package

Title Clusterized and parallelized implmentation of DECO algorithm

Version 0.1.0

Author J. Carter, S. Davenport, J. Knoblauch, G. Morina

Maintainer Sam Davenport <sam.davenport@spc.ox.ac.uk>

Description The package provides methods to run the DECO algorithm as described in ``DECOrrelated feature space partitioning
for distributed sparse regression" in Wang, Dunson, and Leng (2016).

License Undefined

LazyData TRUE

Imports Rcpp (>= 0.12.8), RcppArmadillo (>= 0.7.500.0)

LinkingTo Rcpp, RcppArmadillo

Depends RcppArmadillo, parallel, glmnet

RoxygenNote 5.0.1

R topics documented:

DECO_LASSO_C *DECO Parallelized Algorithm (Pure C)*

Description

This function is deprecated. Use DECO_LASSO_C_PARALLEL function.

Usage

```
DECO_LASSO_C(Y, X, p, n, lambda, r, ncores = 1L, intercept = TRUE)
```

Details

This function is equivalent to DECO_LASSO_C_PARALLEL function when fixing m=1, ncores=1.

DECO_LASSO_C_PARALLEL *DECO Parallelized Algorithm (Pure C++)*

Description

This implements the algorithm DECO which was introducted in "DECOrrelated feature space partitioning for distributed sparse regression" by Wang, Dunson, and Leng (2016). It assumes that we take the lasso to be the penalized regression scehme.

Usage

```
DECO_LASSO_C_PARALLEL(Y, X, p, n, m, lambda, r_1, r_2 = 0.01, ncores = 1L,
  intercept = TRUE, refinement = TRUE, glmnet = TRUE,
  parallel_glmnet = FALSE, precision = 1e-07, max_iter = 100000L)
```

Arguments

Y	gives the nx1 vector of observations we wish to approximate with a linear model of type Y = Xb + e.
X	gives the nxp matrix of regressors, each column corresponding to a different regressor.
p	is the column dimension of X [equivalently, p is the number of regressor variables]. If not given, it is computed as the number of columns of X.
n	is the row dimension of X (and Y) [equivalently, n is the number of observations/individuals]. If not specified, it is computed as the number of rows of X.
m	is the number of groups/blocks you wish to split X into, denoted X(i) for 1 <= i <= m.
lambda	gives the (fixed) penalty magnitude in the LASSO fit of the algorithm.
r_1	is a tweaking parameter for making the inverse more robust (as we take inverse of XX + r_1*I).
r_2	is a tweaking parameter for making the inverse more robust (as we take inverse of X_MX_M + r_2*I).

ncores	determines the number of cores used on each machine to parallelize computation.
intercept	determines whether to include an intercept in the model or not.
refinement	determines whether to include the refinement step (Stage 3 of the algorithm).
glmnet	determines whether glmnet function form glmnet R package should be used to compute the Lasso coefficients. See details for further information. If set to FALSE, C++ implementation of coordinate descent algorithm is used.
parallel_glmnet	determines whether a parallel version of the Lasso coefficients should be used. This parameter is ignored when glmnet is set to FALSE (see details).
precision	determines the precision used in the coordinate descent algorithm. It is ignored when glmnet is set to TRUE.
max_iter	determines the maximum number of iterations used in the coordinate descent algorithm. It is ignored when glmnet is set to TRUE.

Details

This function is a C++ implementation of DECO_LASSO_R and DECO_LASSO_MIX functions. Due to the fact that it is entirely written in C++ it runs faster than the corresponding R implementations for sufficiently large matrices.

Two functions can be used to compute Lasso coefficients: glmnet R function (glmnet = TRUE). and coordinate descent algorithm (glmnet = FALSE). glmnet R function is generally faster, but more memory is required to pass the input argumentd from C++ to R and back. When parallel_glmnet = TRUE an R parallelized version of glmnet is used. Note however that for small datasets this could lead to slower run times, due to the communication between C++ and R.

Descent coordinate algorithm is always run in a parallel way (using ncores threads).

Value

An estimate of the coefficients b.

Author(s)

Samuel Davenport, Jack Carter, Giulio Morina, Jeremias Knoblauch

---

DECO_LASSO_MIX *DECO Parallelized Algorithm (Mixture of R and C++)*

Description

DECO Parallelized Algorithm (Mixture of R and C++)

Usage

```
DECO_LASSO_MIX(Y, X, p = NULL, n = NULL, m = 1, lambda, r_1, r_2 = r_1,
  ncores = 1, intercept = TRUE, refinement = TRUE)
```

Arguments

Y	gives the nx1 vector of observations we wish to approximate with a linear model of type Y = Xb + e
X	gives the nxp matrix of regressors, each column corresponding to a different regressor
p	is the column dimension of X [equivalently, p is the number of regressor variables]. If not given, it is computed as the number of columns of X.
n	is the row dimension of X (and Y) [equivalently, n is the number of observations/individuals] If not given, it is computed as the number of rows of X.
m	is the number of groups/blocks you wish to split X into, denoted X(i) for 1 <= i <= m
lambda	gives the (fixed) penalty magnitude in the LASSO fit of the algorithm
r_1	is a tweaking parameter for making the inverse more robust (as we take inverse of XX + r_1*I)
r_2	is a tweaking parameter for making the inverse more robust (as we take inverse of X_MX_M + r_2*I)
ncores	determines the number of cores used on each machine to parallelize computation
intercept	determines whether to include an intercept in the model or not
refinement	determines whether to include the refinement step (Stage 3 of the algorithm)

Details

The algorithm is based on the description in "DECOrrelated feature space partitioning for distributed sparse regression" in Wang, Dunson, and Leng (2016) if lambda is fixed and LASSO is used as the penalized regression scheme.

Note

-This implementation uses both R functions and C++ functions. In particular, `standardizeMatrix`, `invSymmMatrix`, `squareRootSymmetric` functions are used when needed in place of native R functions. Higher speed can be achieved by using other functions provided in the package.

-This implementation is suboptimal in that X is already stored in the memory when we start the procedure. Ideally, one would give in only the LOCATION X is stored at and read it in chunkwise (thus allowing for larger matrices X, as was intended by the authors).

Author(s)

Samuel Davenport, Jack Carter, Giulio Morina, Jeremias Knoblauch

---

DECO_LASSO_R *DECO Parallelized Algorithm (Pure R)*

Description

DECO Parallelized Algorithm (Pure R)

Usage

```
DECO_LASSO_R(Y, X, p = NULL, n = NULL, m = 1, lambda, r_1, r_2 = r_1,
  ncores = 1, intercept = TRUE, refinement = TRUE)
```

Arguments

Y	gives the nx1 vector of observations we wish to approximate with a linear model of type Y = Xb + e
X	gives the nxp matrix of regressors, each column corresponding to a different regressor
p	is the column dimension of X [equivalently, p is the number of regressor variables]. If not given, it is computed as the number of columns of X.
n	is the row dimension of X (and Y) [equivalently, n is the number of observations/individuals] If not given, it is computed as the number of rows of X.
m	is the number of groups/blocks you wish to split X into, denoted X(i) for 1 <= i <= m
lambda	gives the (fixed) penalty magnitude in the LASSO fit of the algorithm
r_1	is a tweaking parameter for making the inverse more robust (as we take inverse of XX + r_1*I)
r_2	is a tweaking parameter for making the inverse more robust (as we take inverse of X_MX_M + r_2*I)
ncores	determines the number of cores used on each machine to parallelize computation
intercept	determines whether to include an intercept in the model or not
refinement	determines whether to include the refinement step (Stage 3 of the algorithm)

Details

The algorithm is based on the description in "DECOrrelated feature space partitioning for distributed sparse regression" in Wang, Dunson, and Leng (2016) if lambda is fixed and LASSO is used as the penalized regression scheme. The rotated versions of Y and X the authors denote with Tilde are denoted as X* and Y* in the comments below

Note

-This implementation uses only R functions. Higher speed can be achieved by using other functions provided in the package.

-This implementation is suboptimal in that X is already stored in the memory when we start the procedure. Ideally, one would give in only the LOCATION X is stored at and read it in chunkwise (thus allowing for larger matrices X, as was intended by the authors).

-The notation #~PARALLEl~# will be introduced in the code whereever one may achieve signficiant gains from parallelizing

-I could evaluate old expressions in the R version within the mcapply loops! ->saves memory as we write over old data

-We cannot disturb variable order within the algorithm for output comparison reasons, thus reorder X columns before running DECO_LASSO (if important)

Author(s)

Samuel Davenport, Jack Carter, Giulio Morina, Jeremias Knoblauch

DECO_LASSO_R_CLUSTER *DECO Clusterized Algorithm (Pure R)*

Description

DECO Clusterized Algorithm (Pure R)

Usage

```
DECO_LASSO_R_CLUSTER(Y, X, p, n, lambda, r_1, clust, r_2 = r_1, ncores = 1,
  intercept = TRUE, refinement = TRUE)
```

Arguments

Y	gives the nx1 vector of observations we wish to approximate with a linear model of type Y = Xb + e
X	gives the nxp matrix of regressors, each column corresponding to a different regressor
p	is the column dimension of X [equivalently, p is the number of regressor variables]. If not given, it is computed as the number of columns of X.
n	is the row dimension of X (and Y) [equivalently, n is the number of observations/individuals] If not given, it is computed as the number of rows of X.
lambda	gives the (fixed) penalty magnitude in the LASSO fit of the algorithm
r_1	is a tweaking parameter for making the inverse more robust (as we take inverse of XX + r_1*I)
clust	an object obtained by makePSOCKcluster
r_2	is a tweaking parameter for making the inverse more robust (as we take inverse of X_MX_M + r_2*I)
ncores	determines the number of cores used on each machine to parallelize computation
intercept	determines whether to include an intercept in the model or not
refinement	determines whether to include the refinement step (Stage 3 of the algorithm)

Details

The algorithm is based on the description in "DECOrrelated feature space partitioning for distributed sparse regression" in Wang, Dunson, and Leng (2016) if lambda is fixed and LASSO is used as the penalized regression scheme. The rotated versions of Y and X the authors denote with Tilde are denoted as X* and Y* in the comments below

Note

-This implementation uses only R functions.

- This implementation is meant to distribute the load of work to several machines. Note that the current implementation does not deal with the problem of storing big matrices; this function is just the starting step and it should be further developed (i.e. reading the matrix chunckwise from a file, C++ implementation,parallelizing on each machine,...).

Author(s)

Samuel Davenport, Jack Carter, Giulio Morina, Jeremias Knoblauch

invSymmMatrix *Inverse of a matrix*

Description

Inverse of a matrix

Usage

```
invSymmMatrix(M)
```

Arguments

M	a symmetric quadratic matrix

Details

To compute the square root of the matrix, `inv_sympd` function of Armadillo library is used.

Value

The inverse of the matrix

Note

This function is about 2.5x times faster than R function `solve`.

Note that no check is done to test if the matrix M is actually symmetric.

Examples

```
require(rbenchmark)
M <- matrix(rnorm(1000^2,10,5), nrow=1000)
M_symm <- M%*%t(M)
benchmark(solve(M_symm),invSymmMatrix(M_symm),order='relative')
```

lassoCoef *Lasso coefficients (singol core implementation)*

Description

Lasso coefficients (singol core implementation)

Usage

```
lassoCoef(X, Y, nlambda = 1, lambda, intercept = FALSE)
```

Arguments

X	gives the nxp matrix of regressors, each column corresponding to a different regressor
Y	gives the nx1 vector of observations we wish to approximate with a linear model of type Y = Xb + e
nlambda	how many lambda should be tested. Right now only nlambda=1 is supported.
lambda	gives the (fixed) penalty magnitude in the LASSO fit of the algorithm
intercept	determines whether to include an intercept in the model or not

Details

This function uses glmnet package to compute Lasso coefficient.

Check lassoCoefParallel for a parallelized version of this function.

Value

The coefficients of the Lasso regression.

Author(s)

Samuel Davenport, Jack Carter, Giulio Morina, Jeremias Knoblauch

---

lassoCoefParallel *Lasso coefficients (parallel implementation)*

Description

Lasso coefficients (parallel implementation)

Usage

```
lassoCoefParallel(data, Y, nlambda = 1, lambda, intercept = FALSE, m)
```

Arguments

data	is a list of m matrices X. Each X gives the nxp matrix of regressors, each column corresponding to a different regressor
Y	gives the nx1 vector of observations we wish to approximate with a linear model of type Y = Xb + e
nlambda	how many lambda should be tested. Right now only nlambda=1 is supported.
lambda	gives the (fixed) penalty magnitude in the LASSO fit of the algorithm
intercept	determines whether to include an intercept in the model or not

Details

This function uses glmnet package to compute Lasso coefficient in a parallel way.

Check lassoCoef for a not parallelized version of this function.

Value

The coefficients of the Lasso regression.

Author(s)

Samuel Davenport, Jack Carter, Giulio Morina, Jeremias Knoblauch

---

`mulMatrices` *Multiply two matrices*

Description

Multiply two matrices

Usage

```
mulMatrices(A, B)
```

Arguments

`A`	a matrix
`B`	a matrix whose size is compatible with the size of A

Details

To compute such product, Armadillo library is used.

Value

The matrix product A*B.

Note

This function takes about the same speed as R matrix multiplication.

Examples

```
require(rbenchmark)
A <- matrix(rnorm(1000*500,10,5), nrow=1000, ncol=500)
B <- matrix(rnorm(1000*500,10,5), nrow=500, ncol=1000)
benchmark(A%*%B,mulMatrices(A,B),order='relative')
```

`squareRootSymmetric` *Square root of a symmetric matrix*

Description

Square root of a symmetric matrix

Usage

```
squareRootSymmetric(M)
```

Arguments

`M`	a symmetric matrix

Details

To compute the square root of the matrix, `sqrtmat_sympd` function of Armadillo library is used.

Value

A matrix C which is the square root matrix of M (i.e. C*C=M)

Note

This function is about 100 times faster than R function `sqrtm` (contained in expm package).

Note that no check is done to test if the matrix M is actually symmetric.

Author(s)

Samuel Davenport, Jack Carter, Giulio Morina, Jeremias Knoblauch

Examples

```
require(expm)
require(rbenchmark)
A <- matrix(rnorm(10000,mean=10,sd=5),nrow=100)
A_symm <- A%*%t(A)
benchmark(sqrtm(A_symm),squareRootSymmetric(A_symm), order='relative')
```

standardizeMatrix *Standardize a matrix so that its mean is equal to 0*

Description

Standardize a matrix so that its mean is equal to 0

Usage

```
standardizeMatrix(M)
```

Arguments

M	a matrix

Details

The matrix is standardized by subtracting each column with the mean of that column.

Value

A matrix with mean 0

Note

In general, this function does not return a matrix with variance equal to 1.

This function is about 2.5x times faster than doing `scale(M,scale=FALSE)`.

Examples

```
require(rbenchmark)
M <- matrix(rnorm(1000*5000,10,5), nrow=1000)
benchmark(scale(M,scale=FALSE),standardizeMatrix(M), order='relative')
```

standardizeVector *Standardize a vector*

Description

Standardize a vector

Usage

```
standardizeVector(V)
```

Arguments

V	a vector

Details

To compute the standardized vector, each of its entries is subtracted with the vector's mean.

Value

Returns a vector with mean equal to 0.

Note

In general, this function does not return a vector with variance equal to 1.

This function is about 2x slower than directly computing `v-mean(v)`, but it is faster than doing `scale(v,scale=FALSE)`

Author(s)

Samuel Davenport, Jack Carter, Giulio Morina, Jeremias Knoblauch

Examples

```
require(rbenchmark)
v <- 1:5000000
benchmark({v-mean(v)},standardizeVector(v),scale(v,scale=FALSE), order='relative')
```

---

tMatrix *Transpose of a matrix*

Description

Transpose of a matrix

Usage

```
tMatrix(M)
```

Arguments

`M`	a matrix

Details

To compute the transpose matrix, Armadillo library is used.

Value

Its transpose

Note

This function is about 4 times slower than R function `t`.

Examples

```
require(rbenchmark)
M <- matrix(rnorm(1000*5000,10,5), nrow=1000)
benchmark(t(M),tMatrix(M),order='relative')
```

Index